ASAP.V2 and ASAP.V3: Sequential optimization of an Algorithm Selector and a Scheduler

François Gonard, Marc Schoenauer, Michèle Sebag

Journal of Machine Learning Research 1–4

ASAP.V2 and ASAP.V3: Sequential optimization of an Algorithm Selector and a Scheduler

François Gonard* FRANCOIS.GONARD@INRIA.FR
IRT SystemX
LRI, CNRS–INRIA–Université Paris-Sud, Université Paris-Saclay

Marc Schoenauer MARC.SCHOENAUER@INRIA.FR
Michèle Sebag MICHELE.SEBAG@INRIA.FR
LRI, CNRS–INRIA–Université Paris-Sud, Université Paris-Saclay

Abstract

Algorithm portfolios are known to offer robust performances, efficiently overcoming the weakness of every single algorithm on some particular problem instances. The presented *ASAP* system relies on the alternate optimization of two complementary portfolio approaches, namely a sequential scheduler and a per-instance algorithm selector.

1. Introduction

Algorithm selection comes in different flavors, depending on whether the goal is to yield optimal performance in expectation with respect to a given distribution of problem instances (global algorithm selection), or an optimal performance on a particular problem instance (*per instance* algorithm selection). The ASAP systems combine a global *pre-scheduler* and a *per instance* algorithm selector, to take advantage of the diversity of the problem instances on one hand and of the algorithms on the other hand (Gonard et al., 2016). Specifically, the pre-scheduler sequentially launches a few algorithms with a small computational budget each, expectedly solving the "easy" problem instances, and hands over the remaining problem instances to the algorithm selector, selecting an algorithm well-suited to the problem instance at hand. Note that the pre-scheduler yields some additional features characterizing the problem instance at hand, which are used together with the initial descriptive features for the training of the algorithm selector module.

2. Overview

Definition 1 (Pre-scheduler) *Let $\mathcal{A}$ be a set of algorithms. A κ-component pre-scheduler is defined as a sequence of κ (algorithm a_i, time-out τ_i) pairs:*

$$((a_i, \tau_i)_{i=1}^{\kappa}) \;\; with\, (a_i, \tau_i) \in \mathcal{A} \times \mathbb{R}^+, \;\; \forall i \in 1, \ldots, \kappa$$

On problem instance $\mathbf{x}$, the pre-scheduler sequentially launches algorithm a_i with time-out τ_i until either a_i solves $\mathbf{x}$, or time τ_i is reached, or a_i stops without solving $\mathbf{x}$. If $\mathbf{x}$ has been solved, the execution stops. Otherwise, i is incremented while $i \leq \kappa$.

* This work has been carried out in the framework of IRT SystemX and therefore granted with public funds within the scope of the French Program "Investissements d'Avenir".

A pre-scheduler is meant to both contribute to better peak performance and increase the overall robustness of the solving process by mitigating the impact of algorithm selection failures (where the selected algorithm requires large computational resources to solve a problem instance or fails to solve it), as it increases the chance for each problem instance to be solved within seconds, everything else being equal. Accordingly, ASAP combines a pre-scheduler aimed at solving as many problem instances as possible in a first stage, and an algorithm selector taking care of the remaining instances. In its last version (Gonard et al., 2016), ASAP tackles the alternate optimization of the pre-scheduler and the algorithm selector modules. Note that both optimization problems are interdependent: the algorithm selector must focus on the problem instances which are not solved by the pre-scheduler, while the pre-scheduler must symmetrically focus on the problem instances which are most uncertain or badly handled by the algorithm selector. Formally, this interdependence is handled sequentially:

1. A pre-scheduler PS_{ini} is built to optimize the number of instances solved over all training problem instances within a small budget;
2. A performance model $\mathcal{G}(\mathbf{x}, a)$ is built for each algorithm over all training problem instances, using the successes and failures of the algorithms in PS_{ini} as additional features. This performance model yields AS_{ini};
3. A fine-tuned pre-scheduler PS_{opt} is built to optimize the joint performance (PS_{opt}, AS_{ini}) over all training problem instances; in comparison to PS_{ini}, PS_{opt} gets little reward at solving instances that AS_{ini} solves quickly;
4. A second performance model $\mathcal{G}_2(\mathbf{x}, a)$ is built for each algorithm over all training problem instances, using additional features derived from PS_{opt}, yielding AS_{opt}.

ASAP.V2 and V3 are composed of PS_{opt} followed by AS_{opt}. Implementation is in Python 3.[1] ASAP.V2 (resp. ASAP.V3) runs in less than 30' (resp. less than 3h) on a single core (Intel Xeon @2.6 GHz) for scenarios containing less than 10,000 instances and 30 algorithms.

ASAP pre-scheduler A first decision regards the division of labor between the two modules: how to split the available runtime between the two, and how many algorithms are involved in the pre-scheduler (parameter κ). In ASAP.V2, the number κ of algorithms in the pre-scheduler is set to 3, following, e.g., Kadioglu et al. (2011) while in ASAP.V3, κ is optimized in $\{1, \ldots, 4\}$ depending on the scenario.

The pre-scheduler computational budget $T_{ps} = \sum_{i=1}^{\kappa} \tau_i$ is subject to a bi-objective optimization problem (the higher T_{ps}, the more instances the pre-scheduler can solve but the less budget remains for the algorithm selector); it is selected from the cumulative performance curve using a knee detection heuristics, as illustrated on Fig. 1.

The small κ value ($\kappa \leq 4$ in the experiments) makes it possible to exhaustively optimize the subset of κ algorithms $(a_1, \ldots, a_\kappa)$ granted that time-outs are set uniformly ($\tau_i = T_{ps}/\kappa,\ i = 1 \ldots \kappa$).

PS_{opt} differs from PS_{ini} as time-outs $(\tau_1, \ldots, \tau_{\kappa-1})$ are optimized using the black-box optimizer CMA-ES (Hansen et al., 2003), to minimize the regularized average PAR10 score[2] of (PS_{opt}, AS_{ini}) subject to $\sum_{i=1}^{\kappa-1} \tau_i \leq T_{ps}$. PS_{opt} optimization differs from the optimization of

1. Available at: https://gitlab.com/francois.gonard/asap-v2-stable
2. The PAR10 score is defined as the runtime to solve the instance, or 10 times the scenario cutoff time for unsolved instances.

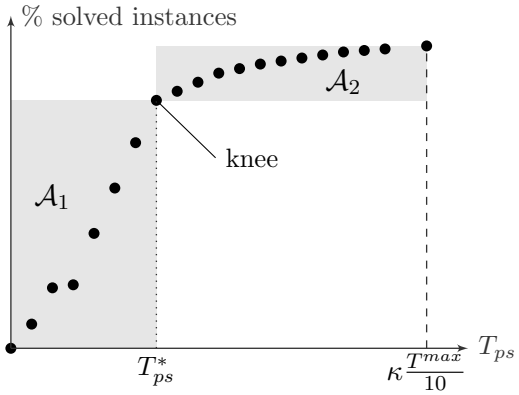

Figure 1: Knee detection on the cumulative performance curve. A somehow optimal trade-off is obtained for T^*_{ps}, minimizing the grey shaded area $\mathcal{A}_1 + \mathcal{A}_2$.

the pre-solving schedule in claspfolio2 (Hoos et al., 2014) in two ways: all training instances are used to fit AS_{ini} while the selector performance is estimated using cross-validation in claspfolio2; to counterbalance this over-optimistic selector and prevent overfitting, the following regularization term is used in ASAP:

$$w \cdot T^{max} \cdot \sqrt{\sum_{i=1}^{\kappa} (\tau_i / T_{ps})^2}$$

with weight $w = 0.005$ (fixed after preliminary cross-validation experiments on the training data) and T^{max} the scenario cutoff time.

ASAP algorithm selector The algorithm selector module relies on a performance model for each algorithm, learned from the training problem instance PAR10 score using a random forest regression algorithm. Random forests are chosen[3] for their robustness against overfitting (Breiman, 2001), which is to be feared considering the small size of some of the scenarios. A main difficulty comes from the representation of problem instances. Typically, feature values are missing for some groups of features for quite a few problem instances, for diverse reasons (computation exceeded time limit, exceeded memory, presolved the instance, crashed, other, unknown). This missing data problem is handled by i) replacing the missing value by the feature average value; ii) adding to the set of descriptive features extra Boolean features, indicating for each feature group whether the values are available or the reason why they are missing otherwise. Eventually the evaluation of the feature relevance in random forests is used to prune the less relevant features.

AS_{ini} (resp. AS_{opt}) is trained using κ additional Boolean features indicating whether the instance is solved by each algorithm of PS_{ini} (resp. PS_{opt}) in the imparted time. All these features take value false for test instances since they are passed to the algorithm selector only if all algorithms in the pre-scheduler failed to solve them.

3. All hyper-parameters are set to their default value in the scikit-learn (Pedregosa, F. et al., 2011) Python library, with 200 trees.

References

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

François Gonard, Marc Schoenauer, and Michèle Sebag. Algorithm selector and prescheduler in the ICON challenge. In *Int. Conf. on Metaheuristics and Nature Inspired Computing (META'2016)*, 2016.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized Evolution Strategy with Covariance Matrix Adaptation. *Evolutionary Computation*, 11(1):1–18, 2003.

Holger Hoos, Marius Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, 14 (4-5):569–585, 2014.

Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Int. Conf. on Principles and Practice of Constraint Programming*, pages 454–469. Springer, 2011.

Pedregosa, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.